Bộ tài liệu hands-on dành cho ReactJS Fresher, bao gồm Unit, Integration, UI và E2E. Mỗi mục đều có **mẫu (pattern) chuẩn doanh nghiệp**, **test case mẫu**, và **lệnh chạy** để bạn áp dụng ngay.

0) Tech stack & cài đặt nhanh

Khuyến nghị (2025): Vite + React + **Vitest** + **React Testing Library** (RTL) + **MSW** (mock API) + **Playwright** (E2E) + **Storybook** (UI)

Bạn có thể thay Vitest bằng Jest nếu team đã chuẩn hóa Jest.

```
# Tạo project
npm create vite@latest react-testing-handson -- --template react-ts
cd react-testing-handson

# Testing cho Unit/Integration/UI
npm i -D vitest @testing-library/react @testing-library/jest-dom @testing-library/user-event \
msw whatwg-fetch

# E2E với Playwright
npm i -D @playwright/test

# Storybook cho UI Testing
npx storybook@latest init

# (Tuỳ chọn) Kiểm tra type cho test
npm i -D @types/testing-library__jest-dom
```

package.json – scripts gợi ý

```
{
  "scripts": {
    "test": "vitest --run",
    "test:ui": "vitest",
    "test:cov": "vitest --coverage",
    "test:e2e": "playwright test",
    "test:e2e:ui": "playwright test --ui",
```

```
    "storybook": "storybook dev -p 6006",
    "build-storybook": "storybook build",
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}
```

vitest.config.ts (cấu hình tối thiểu)

```
import { defineConfig } from 'vitest/config'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  test: {
    environment: 'jsdom',
    setupFiles: './test/setupTests.ts',
    css: true,
    coverage: {
      reporter: ['text', 'lcov'],
      lines: 80, functions: 80, branches: 70, statements: 80
    }
  }
})
```

test/setupTests.ts

```
import '@testing-library/jest-dom'
import { server } from './testServer'
// MSW: mock API
beforeAll(() => server.listen({ onUnhandledRequest: 'error' }))
afterEach(() => server.resetHandlers())
afterAll(() => server.close())
```

test/testServer.ts (MSW)

```
import { setupServer } from 'msw/node'
import { http, HttpResponse } from 'msw'

export const handlers = [
  http.get('/api/me', () => HttpResponse.json({ id: 1, name: 'Alice' })),
  http.post('/api/login', async ({ request }) => {
    const body = await request.json()
    if (body.username === 'bob' && body.password === '123') {
      return HttpResponse.json({ token: 'jwt.token.sample' })
    }
    return new HttpResponse('Unauthorized', { status: 401 })
  })
]

export const server = setupServer(...handlers)
```

Cấu trúc thư mục gợi ý

```
src/
  app/              # providers, store
  components/       # UI components
  features/         # mỗi feature: component + slice + hooks + tests
  hooks/
  pages/
  services/         # api clients
  utils/
test/
  setupTests.ts
  testServer.ts
```

1) Chuẩn doanh nghiệp: Naming, Pattern & Convention

- **Tên file test:** *.test.tsx hoặc *.spec.tsx (theo team).
- **Tên test case:** action_expected_condition
  - Ví dụ: submit_shouldShowError_whenPasswordEmpty
- **Mẫu AAA (Arrange-Act-Assert)** rõ ràng; **Given-When-Then** khi cần behavior.
- **Selector ưu tiên** getByRole, getByLabelText, getByText → tránh data-testid trừ khi bắt buộc.
- **Test nhỏ, độc lập**, không phụ thuộc thứ tự chạy, **không chia sẻ state**.
- **Mock boundary**, không mock logic nội bộ:
  - Mock **HTTP** bằng **MSW**, **time** bằng vi.useFakeTimers(), **Date/UUID** bằng injection.
- **Coverage target:** Lines/Funcs/Statements ≥ 80%, Branches ≥ 70% (điều chỉnh theo risk).
- **Pyramid:** Unit (nhiều) > Integration (vừa) > E2E (ít nhưng critical).

2) Unit Testing (logic thuần, hooks đơn lẻ)

2.1 Test hàm utils (pure function)

src/utils/calc.ts

```
export const addTax = (price: number, rate: number) => {
  if (rate < 0) throw new Error('Invalid rate')
  return +(price * (1 + rate)).toFixed(2)
}
```

src/utils/calc.test.ts

```
import { describe, it, expect } from 'vitest'
import { addTax } from './calc'

describe('addTax', () => {
  it('returnsCorrectTotal_whenRateValid', () => {
    expect(addTax(100, 0.1)).toBe(110)
  })
  it('throwsError_whenRateNegative', () => {
    expect(() => addTax(100, -0.1)).toThrow('Invalid rate')
  })
})
```

2.2 Test custom hook (debounce)

src/hooks/useDebounce.ts

```
import { useEffect, useState } from 'react'
export function useDebounce<T>(value: T, delay = 300) {
  const [debounced, setDebounced] = useState(value)
  useEffect(() => {
    const id = setTimeout(() => setDebounced(value), delay)
    return () => clearTimeout(id)
  }, [value, delay])
  return debounced
}
```

src/hooks/useDebounce.test.tsx

```
import { describe, it, expect, vi } from 'vitest'
import { renderHook, act } from '@testing-library/react'
import { useDebounce } from './useDebounce'

describe('useDebounce', () => {
  it('updatesValue_afterDelay', () => {
    vi.useFakeTimers()
    const { result, rerender } = renderHook(({ v }) => useDebounce(v, 500), {
      initialProps: { v: 'a' }
    })
    expect(result.current).toBe('a')
    rerender({ v: 'ab' })
    expect(result.current).toBe('a')
    act(() => { vi.advanceTimersByTime(500) })
    expect(result.current).toBe('ab')
    vi.useRealTimers()
  })
})
```

3) Component Testing (RTL) – UI tương tác cơ bản

src/components/LoginForm.tsx

```
import { useState } from 'react'

type Props = { onSubmit: (u: string, p: string) => Promise<void> }
export default function LoginForm({ onSubmit }: Props) {
  const [loading, setLoading] = useState(false)
  const [error, setError] = useState<string | null>(null)

  const handle = async (e: React.FormEvent) => {
    e.preventDefault()
    const form = e.target as HTMLFormElement
    const u = (form.elements.namedItem('username') as HTMLInputElement).value
    const p = (form.elements.namedItem('password') as HTMLInputElement).value
    setError(null)
    setLoading(true)
    try { await onSubmit(u, p) }
    catch (e: any) { setError(e.message ?? 'Login failed') }
    finally { setLoading(false) }
  }

  return (
    <form onSubmit={handle}>
      <label htmlFor="username">Username</label>
      <input id="username" name="username" />
      <label htmlFor="password">Password</label>
      <input id="password" name="password" type="password" />
      {error && <div role="alert">{error}</div>}
      <button type="submit" disabled={loading}>
        {loading ? 'Logging in…' : 'Login'}
      </button>
    </form>
  )
}
```

src/components/LoginForm.test.tsx

```
import { render, screen, within } from '@testing-library/react'
import userEvent from '@testing-library/user-event'
import { describe, it, expect } from 'vitest'
import LoginForm from './LoginForm'

describe('LoginForm', () => {
  it('submit_shouldCallOnSubmit_withFormValues', async () => {
    const user = userEvent.setup()
    const onSubmit = vi.fn().mockResolvedValue(undefined)
    render(<LoginForm onSubmit={onSubmit} />)

    await user.type(screen.getByLabelText(/username/i), 'bob')
    await user.type(screen.getByLabelText(/password/i), '123')

    await user.click(screen.getByRole('button', { name: /login/i }))
    expect(onSubmit).toHaveBeenCalledWith('bob', '123')
  })

  it('showsError_whenOnSubmitRejects', async () => {
    const user = userEvent.setup()
    const onSubmit = vi.fn().mockRejectedValue(new Error('Bad creds'))
    render(<LoginForm onSubmit={onSubmit} />)

    await user.type(screen.getByLabelText(/username/i), 'bob')
    await user.type(screen.getByLabelText(/password/i), 'wrong')
    await user.click(screen.getByRole('button', { name: /login/i }))

    expect(await screen.findByRole('alert')).toHaveTextContent('Bad creds')
    // Button re-enabled after fail
    const btn = screen.getByRole('button', { name: /login/i })
    expect(btn).not.toBeDisabled()
  })
})
```

Pattern chính:

- Sử dụng **role & label** thay vì data-testid.
- Test **state loading** và **error path**.
- Không kiểm tra implementation details (state nội bộ), chỉ assert **UI & behavior**.

4) Integration Testing (nhiều phần ghép lại)

Ví dụ: Component + Redux store + API mock (MSW).

src/app/store.ts

```
import { configureStore, createSlice, createAsyncThunk } from '@reduxjs/toolkit'

export const fetchMe = createAsyncThunk('auth/fetchMe', async () => {
  const res = await fetch('/api/me'); return res.json()
})

const auth = createSlice({
  name: 'auth',
  initialState: { user: null as null | { id: number; name: string }, loading: false },
  reducers: {},
  extraReducers: (b) => {
    b.addCase(fetchMe.pending, (s) => { s.loading = true })
     .addCase(fetchMe.fulfilled, (s, a) => { s.user = a.payload; s.loading = false })
  }
})

export const store = configureStore({ reducer: { auth: auth.reducer } })
export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch
```

src/features/auth/Profile.tsx

```
import { useEffect } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import type { RootState, AppDispatch } from '../../app/store'
import { fetchMe } from '../../app/store'

export default function Profile() {
  const dispatch = useDispatch<AppDispatch>()
  const { user, loading } = useSelector((s: RootState) => s.auth)

  useEffect(() => { dispatch(fetchMe()) }, [dispatch])

  if (loading) return <p>Loading…</p>
  if (!user) return <p>No user</p>
  return <h2 aria-label="greeting">Hello, {user.name}</h2>
}
```

src/features/auth/Profile.int.test.tsx

```
import { render, screen } from '@testing-library/react'
import { Provider } from 'react-redux'
import { store } from '../../app/store'
import Profile from './Profile'

describe('Profile (Integration)', () => {
  it('loadsAndShowsUser_whenApiOk', async () => {
    render(<Provider store={store}><Profile/></Provider>)
    expect(screen.getByText(/loading/i)).toBeInTheDocument()
    const greet = await screen.findByLabelText('greeting')
    expect(greet).toHaveTextContent('Hello, Alice')
  })
})
```

Pattern chính:

- Không mock Redux store trừ khi cần; dùng store thật cho Integration.
- API đã **mock** bằng **MSW** → kiểm thử contract UI <-> API.

5) UI Testing (Storybook + Storybook Test Runner / Playwright Visual)

5.1 Story cho component (dễ review & docs)

src/components/LoginForm.stories.tsx

```
import type { Meta, StoryObj } from '@storybook/react'
import LoginForm from './LoginForm'

const meta: Meta<typeof LoginForm> = {
  title: 'Auth/LoginForm',
  component: LoginForm
}
export default meta
type Story = StoryObj<typeof LoginForm>

export const Default: Story = {
  args: { onSubmit: async () => {} }
}
```

Chạy:

```
npm run storybook
```

5.2 UI test (interaction) bằng @storybook/test (tuỳ chọn)

npm i -D @storybook/test @storybook/jest @testing-library/dom

Tạo LoginForm.stories.test.ts để chạy cùng Storybook test runner (tham khảo docs Storybook). Hoặc dùng **Playwright** để **visual regression**:

tests/visual/login.spec.ts (Playwright)

```
import { test, expect } from '@playwright/test'

test('LoginForm visual', async ({ page }) => {
  await page.goto('http://localhost:6006/?path=/story/auth-loginform--default')
  await expect(page).toHaveScreenshot('loginform.png', { animations: 'disabled' })
})
```

Khi chạy visual test, bật storybook và dùng toHaveScreenshot để phát hiện lệch UI.

6) E2E Testing (Playwright)

playwright.config.ts

```
import { defineConfig, devices } from '@playwright/test'
export default defineConfig({
  testDir: 'e2e',
  webServer: {
    command: 'npm run dev',
    url: 'http://localhost:5173',
    reuseExistingServer: !process.env.CI
  },
  use: { baseURL: 'http://localhost:5173', headless: true },
  projects: [{ name: 'chromium', use: { ...devices['Desktop Chrome'] } }]
})
```

e2e/login.spec.ts

```
import { test, expect } from '@playwright/test'

test('login_returnsToken_andRedirects_whenValid', async ({ page }) => {
  await page.goto('/')
  await page.getByLabel('Username').fill('bob')
  await page.getByLabel('Password').fill('123')
  await page.getByRole('button', { name: /login/i }).click()

  // Ví dụ: sau khi login thành công, có text "Welcome"
  await expect(page.getByText(/welcome/i)).toBeVisible()
})

test('login_showsError_whenInvalid', async ({ page }) => {
  await page.goto('/')
  await page.getByLabel('Username').fill('bob')
  await page.getByLabel('Password').fill('wrong')
  await page.getByRole('button', { name: /login/i }).click()
  await expect(page.getByRole('alert')).toHaveText(/unauthorized|failed/i)
})
```

Best practice:

- E2E ít nhưng bao phủ flow **xuyên suốt** (login, checkout, form chính).
- Dùng selector theo **role/label/text** để ổn định.
- Có thể mock network ở **API layer** (Playwright route) khi môi trường backend chưa sẵn.

7) Mocking nâng cao (MSW, time, fetch)

Mock 401 trong test cụ thể

```
import { server } from '../../test/testServer'
import { http, HttpResponse } from 'msw'
server.use(
  http.post('/api/login', () => new HttpResponse('Unauthorized', { status: 401 }))
)
```

Mock timer

```
vi.useFakeTimers()
// ... code chạy debounce, timeout
vi.advanceTimersByTime(1000)
vi.useRealTimers()
```

whatwg-fetch đã cài để có fetch trong jsdom.

8) Chuẩn viết test theo "mẫu doanh nghiệp"

1. **Tên method**: action_expected_condition
2. **AAA rõ ràng**:

```
// Arrange
// Act
// Assert
```

3. **Scope nhỏ** (Unit) trước, **ghép dần** (Integration), chỉ **ít E2E**.
4. **Không test implementation detail**, test **hành vi** & **UI hiển thị**.
5. **Giả lập người dùng** bằng userEvent (typing, click, tab…).
6. **Chọn selector chuẩn**: role/label/text, hạn chế data-testid.
7. **Xử lý async**: findBy*, await waitFor(() => …) khi cần.
8. **Giới hạn mock**: mock ngoài ranh giới (HTTP, time, storage).
9. **Dọn dẹp**: server.resetHandlers(), timers real lại sau test.
10. **Coverage & chất lượng**: chạy npm run test:cov trong CI.

9) CI mẫu (GitHub Actions)

.github/workflows/test.yml

```
name: CI
on:
  push: { branches: [ main ] }
  pull_request: { branches: [ main ] }

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '20' }
      - run: npm ci
      - run: npm run test:cov
      - run: npx playwright install --with-deps
      - run: npm run test:e2e
```

10) Checklist "đúng chuẩn" cho Fresher

- **Unit**: utils/hook đã có test case happy + edge (error, boundary).
- **Component**: test input, validation message, loading, error, focus order.
- **Integration**: ghép Store + API mock, đảm bảo UI phản ứng theo state.
- **UI/Storybook**: có story cho các state (default/loading/error/empty).
- **E2E**: 1–3 flow quan trọng (login, tạo/sửa item, checkout).
- **Mock**: HTTP bằng MSW, timer/Date nếu có debounce/expiry.
- **Selectors**: role/label/text – pass với screen reader.
- **Coverage**: đạt ngưỡng team (ví dụ Lines ≥ 80%).
- **Scripts**: test, test:cov, test:e2e chạy OK trên CI.
- **Tài liệu**: README mô tả cách chạy test local & CI.

11) Bài tập Hands-on (giao ngay cho Fresher)

1. **Unit:** Viết test cho addTax, bổ sung case 0, giá lớn, rate=0.08.
2. **Hook:** Sửa useDebounce nhận leading flag (bắn giá trị ngay lần đầu) và viết test cho 2 mode leading: true/false.
3. **Component:** Thêm validate bắt buộc username & password, hiển thị lỗi từng field. Viết test:
   - submit_shouldShowFieldErrors_whenMissingInputs
   - submit_shouldDisableButton_whenSubmitting
4. **Integration:** Tạo TodoList dùng store Redux + API /api/todos. Viết test hiển thị empty state, render todos, toggle done.
5. **UI/Storybook:** Tạo stories cho TodoItem với các state default/done/hover. Viết visual test Playwright so sánh snapshot.
6. **E2E:** Flow “login → tạo todo → đánh dấu hoàn thành → tìm kiếm”. Kịch bản:
   - Điền form login (hợp lệ)
   - Điều hướng đến /todos
   - Thêm item “Study testing”
   - Đánh dấu completed
   - Tìm kiếm “Study” thấy kết quả

12) Bonus: Mẫu test form validation với Zod + react-hook-form

src/components/RegisterForm.tsx

```
import { useForm } from 'react-hook-form'
import { z } from 'zod'
import { zodResolver } from '@hookform/resolvers/zod'

const schema = z.object({
  email: z.string().email(),
  password: z.string().min(6),
  confirm: z.string()
}).refine(d => d.password === d.confirm, { path: ['confirm'], message: 'Passwords must match' })

type Form = z.infer<typeof schema>

export default function RegisterForm({ onSubmit }: { onSubmit: (f: Form)=>void }) {
```

```
  const { register, handleSubmit, formState: { errors, isSubmitting } } =
    useForm<Form>({ resolver: zodResolver(schema) })

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input aria-label="Email" {...register('email')} />
      {errors.email && <span role="alert">{errors.email.message}</span>}
      <input aria-label="Password" type="password" {...register('password')} />
      {errors.password && <span role="alert">{errors.password.message}</span>}
      <input aria-label="Confirm" type="password" {...register('confirm')} />
      {errors.confirm && <span role="alert">{errors.confirm.message}</span>}
      <button disabled={isSubmitting}>Create account</button>
    </form>
  )
}
```

src/components/RegisterForm.test.tsx

```
import { render, screen } from '@testing-library/react'
import userEvent from '@testing-library/user-event'
import RegisterForm from './RegisterForm'

it('showsFieldErrors_whenInvalid', async () => {
  const user = userEvent.setup()
  const onSubmit = vi.fn()
  render(<RegisterForm onSubmit={onSubmit} />)

  await user.click(screen.getByRole('button', { name: /create account/i }))
  expect(await screen.findAllByRole('alert')).toHaveLength(2) // email + password
})

it('showsMatchError_whenConfirmMismatch', async () => {
  const user = userEvent.setup()
  render(<RegisterForm onSubmit={vi.fn()} />)
```

```
  await user.type(screen.getByLabelText(/email/i), 'a@b.com')
  await user.type(screen.getByLabelText(/password/i), '123456')
  await user.type(screen.getByLabelText(/confirm/i), '654321')
  await user.click(screen.getByRole('button', { name: /create account/i }))

  expect(await screen.findByRole('alert')).toHaveTextContent(/must match/i)
})
```

13) Khi nào dùng Jest thay Vitest?

- Team kế thừa **Jest** từ codebase cũ → giữ Jest để giảm xáo trộn.
- Với Vite/React mới → **Vitest** khởi chạy nhanh, cấu hình tối giản, tương thích RTL tốt.